__/__/____

Expt no.	Experiment	Date	Sign
1)	a) Linear Search b) Binary Search c) Selection Sort d) Bubble Sort	31/5/24	10 3/5/24
2)	OUTPUTS: TOPO_SR, DFS, GCD	31/5/24	10 31/5/24
3)	a) Quick Sort b) Merge Sort	7/6/24	10 7/6/24
4)	a) Warshall b) Floyd c) Johnson-trotter d) Knapsack	13/6/24	10 13/6/24
5)	a) Horspool b) HeapSort	21/6/24	10 21/6/24
6)	a) Prim's b) Kruskall c) Dijkstra's d) Knapsack	~~28/6/24~~	9 15/7/24
7)	a) N-queens	~~4/6/24~~	10 15/7/24

__/__/____

Linear Search:

```
#include <stdio.h>
#include <stdlib.h>
void linear (int key, int length, int a[])
{
    for (int p=0; p<length; p++)
    {
        if (a[p]==key)
        {
            printf ("Element is present");
            exit(0);
        }
    }
    printf ("Element is not present");
}
int main ()
{
    int a[100], n, key;
    printf ("Enter the size of the array");
    scanf ("%d", &n);
    printf ("Enter Array:");
    for(int p=0; p<n; p++)
    { scanf("%d", &a[p]); }
    printf ("Enter Key:");
    scanf("%d", &key);
    linear (key, n, a);
    return 0;
}
```

__/__/____

Binary search:

```
#include <stdio.h>
#include<conio.h>
int binary (int key, int length, int a[])
{
   int left = 0;
   int right = len-1;
   while (left <= right)
   {
      int mid= left + (right -(left)/2;
      if (a[mid]==key);
            return a[mid];
      if [a[mid]<key)
            left = mid+1;
      else
            right = mid-1;
   }
   return 0;
}
int main()
{
   int a[100], n, key, found;
   printf (" Enter the size and key");
   scanf ("%d %d", &n, &key);
   printf (" Enter the sorted array:");
   for (int i=0; i<n; i++)
         scanf ("%d", &a[i]);
   found = binary (key, n, a);
   if (found!=0)
         printf ("Element found");
   else
         printf(" Element not found");
   return 0;
}
```

__/__/__

Linear Search O/P:

Case 1
Enter the size of the array: 5
Enter the array elements: 1 2 3 4 5
Enter the key: 3
Element is present.

Case 2
Enter the size of the array: 5
Enter the array elements: 1 2 3 4 5
Enter the key: 6
Element is not present.

Binary search o/p:

Case 1
Enter the size & key of the array:
3 3
Enter the array elements: 1 2 3
The element is present.

Case 2
Enter the size & key of the array:
3 4
Enter the array elements: 1 2 3
The element is not present.

Time complexity of linear search = O(N)

Time complexity of binary search = O(log n)

__/__/____

Selection sort:

```
#Include<stdio.h>
#Include<stdlib.h>
void sort(int a[], int n)
{
    int i, j, temp;
    for(i=0; i<n-1; i++)
    {
        for(j=i+1; j<n; j++)
        {
            if(a[i] > a[j])
            {
                temp = a[i]
                a[i] = a[j]
                a[j] = temp;
            }
        }
    }
int main()
{
    int a[100], n;
    printf("Enter Array size:");
    scanf("%d", &n);
    printf("Enter Array elements:");
    for(int i=0; i<n; i++)
        scanf("%d", &a[i]);
    sort(a, n);
    printf("sorted Array:");
    for(int i=0; i<n; i++)
        printf("%d", a[i]);
    return 0;
}
```

__/__/____

O/P:

Enter the Array size: 5
Enter Array elements: 5 4 3 2 1
Sorted Array: 1 2 3 4 5

Bubble sort:

```
#include<stdio.h>
#include<stdlib.h>
void bubble(int a[], int n)
{ int i, j, temp;
  for(i=0; i<n-1; i++)
  {
    for(j=0; j<n-i-1; j++)
    {
      if(a[j] > a[j+1])
      {
        temp = a[j];
        a[j] = a[j+1];
        a[j+1] = temp;
      }
    }
  }
int main()
{
  int a[100], n;
  printf("Enter Array size: ");
  scanf("%d", &n);
  printf("Enter Array: ");
  for(int i=0; i<n; i++)
    scanf("%d", &a[i]);
  bubble(a, n); printf("sorted Array: ");
  for(int i=0; i<n; i++)
```

__/__/____

```
    printf("%d", a[i]);
  return 0;
}
```

O/P:

Enter the Array size : 5
Enter Array : 5 4 3 2 1
Sorted Array : 1 2 3 4 5

Linear search:
Time complexity = O(N)
Best case = Element is at the first pos.
Worst case = Key is in the last index
Avg case = O(N)

Binary search:
Time complexity = O(log N)
Best case = when element is present at the middle index.
Worst case = when element is at the first position
Avg case = O(log N)

Selection sort:
Time complexity = O(n^2)
Best case = O(n^2) occurs when the element is sorted
Worst case = Array is in descending order
Avg case = O(n^2)

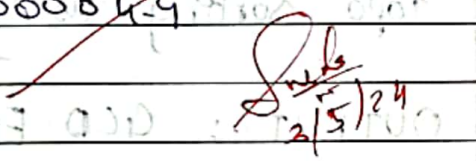
Bubble sort:

Time complexity = O(n)
Best case = O(n), Array is already sorted.
worst case = $O(N^2)$, Array is in decreasing order
Avg case = $O(N^2)$., No. of comparisons is same.

LS
Total execution time = 0.000126

LS & selection sort
total time - 0.000031

Bubble sort
time taken = 0.000049



31/5/24

LAB - 2

→ OUTPUT: TOPO_DFS

Enter vertices = 3
Enter adjacency matrix:
1 1 0
1 0 1
0 1 1
Topo sorting using DFS: 1 0 2

→ OUTPUT: TOPO - SR
Enter vertices: 3
Enter adjacency matrix:
1 1 0
1 0 1
0 1 1
Topo sorting using source reduction: 1 0 2

→ OUTPUT: GCD Euclidean
1ST number = 24
2nd number = 18
Result = 6.

→ OUTPUT: Computing Median
Enter the size of the array = 5
Enter the array elements = 10 8 4 5
Enter the value of k = 3
The 3 smallest element is: 5

→ OUTPUT: TOH
Move disk 1 from A to B

~~LAB - 2~~

Move 1 disk from A to C.
Move 1 disk from B to C.

LAB - 3 7106124

1) Quick sort

```
#include <stdio.h>
void swap (int *a, int *b) {
    int temp = *a;
    *a = *b;
    *b = temp;
}
int partition (int a[], int low, int high){
    int pivot = a[high];
    int i = (low - 1)

    for (int j = low; j <= high-1; j++){
        if (a[j] <= pivot){
            i++;
            swap (&a[i], &a[j]);
        }
    }
    swap (&a[i+1], &a[high]);
    return (i+1);
}

void quicksort( int a; int low, int high){
    if (low < high){
        int pi = partition (a, low, high);
        quick sort (a, low, high);
        quick sort (a, mid+1, high);
    }
}
```

__/__/____

```
void printArray(int a, int size){
    for(int i=0; i<size; i++)
        printf("%d ", a[i]);
    printf("\n");
}
int main(){
    int arr = {10, 7, 8, 9, 1, 5};
    int n = sizeof(arr)/sizeof(arr[0]);
    printf("Given Array: \n");
    printarray(arr, n);
    quick_sort(arr, 0, n-1);
    printf("Sorted Array: \n");
    printArray(arr, n);
    return 0;
}
```

OUTPUT:

Given Array:
10 7 8 9 1 5

Sorted Array:
1 5 7 8 9 10.

→ Merge Sort:

```
#include<stdio.h>
void merge(int a, int l, int m, int r){
    int i, j, k;
    int n1 = m-l+1;
    int n2 = r-m;
    for(i=0; i<n1; i++)
        L[i] = ar.(l+i);
    for(j=0; j<n2; j++)
        R[j] = a.(m+1+j);
    i=0; j=0; k=l;
```

```
while (i<n1 && j<n2) {
    if (L[i] <= R[j]) {
        arr[k] = L[i];
        i++;
    } else {
        a[k] = R[j];
        j++;
    }
    k++;
}
while (i<n1) {
    a[k] = L[i];
    i++;
    k++;
}
while (j<n2) {
    a[k] = L[j];
    j++;
    k++;
}
}
void merge_sort(int a, int l, int r){
    if (l<r) {
        int m = l+(r-1)/2;
        mergesort(a, l, m);
        merge_sort(a, m+1, r);
        merge(a, l, m, r);
    }
}
void printArray(int a[], int size){
    for (int i = 0; i<size; i++)
        printf("%d ", a[i]);
    printf("\n");
}
```

__/__/__

```
int main(){
    int arr[] = {12, 11, 13, 5, 6, 7};
    int arr_size = 6;
    merge_sort(arr, 0, arr_size - 1);
    printf("Sorted Array ps\n");
    printArray(arr, arr_size);
    return 0;
}
```

OUTPUT:

Given Array ps:
12 11 13 5 6 7

Sorted Array ps:
5 6 7 11 12 13.

P.T.O.

13/06/24

LAB - 4

```
#Pnclude<stdPO.h>
VoPd warshall (Pnt a[][10], Pnt n) {
    Pnt p[q[10];
    for(Pnt P= 0; P<n ; P++) {
        for (Pnt ƒ=0; ƒ<n; ƒ++) {
            p[P][ƒ] = a[P][ƒ];
        }
    }
    for(Pnt R= 0; K<n ; k++) {
        for(Pnt P = 0; P<n ; P++) {
            for (Pnt ƒ =0; ƒ<n; ƒ++) {
                Pf (p[P][k] == 1 && p[k][ƒ]==1) {
                    p[P][ƒ] = 1;
                }
            }
        }
    }
    prPntf(" TransPRe closuRe :\n");
    for (Pnt P = 0; P<n ; P++) {
        for (Pnt ƒ= 0; ƒ<n ; ƒ++) {
            prPntf("%d ", p[P][ƒ]);
        }
        pPPntf("\n");
    }
}
Pnt main() {
    Pnt n;
    prPntf(" Enter the number of verPces: ");
    Scanf("%d", &n);
    Pnt a[10][10];
    prPntf(" Enter the adjaency matPx :\n");
```

__/__/____

```
        for (int i = 0; i<n; i++){
            for(int j=0; j<n; j++){
                scanf("%d", &a[i][j]);
            }
        }
        warshall(a,n);
        return 0;
}
```

OUTPUT:

Enter the adjacency matrix:

1 1 0
1 0 1
1 1 0

Transitive closure:

1 1 1
1 1 1
1 1 1

→ Floyd's Algorithm:

```
#include <stdio.h>
#define V 4
#define INF 9999
void printSol(int dist[][V])
{
    printf("The following matrix shows the
        shortest distances between every pair
        of vertices.\n");
    for (int i=0; i<V; i++) {
        for (int j=0; j<V; j++){
            if (dist[i][j] == INF)
```

```
            printf("%7s", "INF");
        else
            printf("%7d", dist[i][j]);
    }
    printf("\n");
}
}
int main()
{
    int graph[V][V] = { { 0, INF, 3, INF },
                        { 2, 0, INF, INF },
                        { INF, 6, 0, 1 },
                        { 7, INF, INF, 0 } };
    floydWarshall(graph);
    return 0;
}
```

OUTPUT :

The following matrix shows the shortest distances between every pair of vertices

```
0   9   3   4
2   0   5   6
8   6   0   1
7   16  10  0
```

→ Johnson Trotter :

```
#include <stdio.h>
#include <stdbool.h>
#define MAX_N 10
int p[MAX_N];
int d[MAX_N];
int n;
```

__/__/____

```
void print_perm() {
    for (int i = 0; i < n; i++)
        printf("%d", p[i]);
    printf("\n");
}
void swap(int *a, int *b) {
    int t = *a;
    *a = *b;
    *b = t;
}
int mobile(int i) {
    if (d[i] == -1 && i != 0)
        return (p[i] > p[i-1]) ? i : -1;
    if (d[i] == 1 && i != n-1)
        return (p[i] > p[i+1]) ? i : -1;
    return -1;
}
void johnson() {
    for (int i = 0; i < n; i++) {
        d[i] = -1;
        p[i] = i+1;
    }
    print_perm();
    bool m = true;
    while (m) {
        m = false;
        int mi = -1;
        int mn = -1;
        for (int i = 0; i < n; i++) {
            int t = mobile(i);
            if (t != -1 && p[i] > mn) {
                mn = p[i];
                mi = t;
```

```
            m=true;
        }
    }
    if(m){
        int mnP;
        if(d[mP]==-1)
            mnP=mP-1;
        else
            mnP=mP+1;
        swap(&P[mnP], &P[mP]);
        swap(&d[mnP], &d[mP]);
        for(int P=0; P<n; P++){
            if(P[P]>mn)
                d[P]=-d[P];
        }
        print perm();
    }
}
int main(){
printf("Enter n:");
scanf("%d",&n);
if(n>MAX_N||n<=0){
    printf("Invalid");
    return 1;
}
johnson();
return 0;
}
```

OUTPUT:

```
Enter n : 2
1 2
2 1
```

13/6/24

__/__/____

→ Knapsack Problem:

```
#include <stdio.h>
int max (int a, int b) {
        return (a > b) ? a : b;
    }
int knapsack (int W, int wt[], int val[], int n)
{
    int i, w;
    int K[n+1][W+1];
    for (i=0; i<=n; i++) {
        for (w=0; w<=W; w++) {
            if (i==0 || w==0)
                K[i][w] = 0;
            else if (wt[i-1] <= w){
                K[i][w] = max (val[i-1] +
            K[i-1] [w- wt[i-1]], K[i-1][w]);
            }
            else
                K[i][w] = K[i-1][w];
        }
    }
    return K[n][W];
}
int main() {
    int val[] = {60, 100, 120}; int wt[] = {10,20,30};
    int W=50; int n=3;
    printf ("Max value = %d \n", knapsack (W, wt,
        val, n));
    return 0;
}
```

O/P:

Max value = 220

21/6/24

LAB - 5:

→ Horspool's Algorithm:

```
#include <stdio.h>
#include <string.h>
#define MAX 128
void shift (char *p, int *shifts) {
    int m = strlen(p);
    for (int i=0 ; i< MAX ; i++)
        shifts[i] = m;
    for (int j= 0; j <m-1 ; j++)
        shifts[(int)p[j]] = m -1- j;
}
int horspool (char *text, char *pattern) {
    int m = strlen(pattern);
    int n = strlen (text);
    int shifts [MAX];
    shift(pattern, shifts);
    int i = m-1;
    while (i<n) {
        int k= 0;
        while (k<m && pattern [m-1-k]
            == text[i-k]) {
            k++;
        } if (k==m)
            return i-m+1;
        else
            i+= shifts [(int) text[i]];
    }
    return -1;
}
int main() {
    char text[] = "JIm saw me in a
        barber shop";
```

__/__/____

```
char pattern[] = "barber";
int pos = horspool(text, pattern);
if (pos != -1)
    printf("Pattern is at %d\n", pos);
else
    printf("Pattern not found\n");
return 0;
}
```

OUTPUT:

Pattern is at position 16

21/11/24

__/__/__

LAB-5

→ Heap Sort (Descending):

Program Implementation:

```
#include<stdio.h>
#include<stdlib.h>
#define MAX 100
void heapify(int a[], int n){
    for(int k=0; k<n; k++){
        int key=a[k];
        int c = k;
        int p = (c-1)/2;
        while(c>0 && key>a[p]){
            a[c] = a[p];
            c=p;
            p = (c-1)/2;
        }
        a[c]= key
    }
}
int main(){
    int arr[MAX], n;
    printf("Enter the size of the binary tree\n");
    scanf("%d", &n);
    printf("Enter the elements of binary tree\n");
    for(int i=0; i<n; i++)
        scanf("%d", &arr[i]);
    heapify(arr, n);
    printf("Heap Sorted Array:\n");
    for(int i=0; i<n; i++)
        printf("%d ", arr[i]);
    printf("END");
    return 0;
}
```

__/__/____

OUTPUT :

Enter the size of the binary tree:
7
Enter the Elements of the binary tree
50 25 30 75 100 45 80

Heap Sorted Array:

100 → 75 → 80 → 25 → 50 → 30 → 45 → END.

LAB - 6

1) Prim's Algorithm:

```
#include<stdio.h>
#include<stdlib.h>
#include<limits.h>
#define V 6
int minKey(int key[], bool mstSet[]){
    int min = INT_MAX
int cost[10][10], n, t[10][2], sum;
void prims(int cost[10][10], int n);
int main(){
    int i, j;
    printf("Enter the number of vertices:");
    scanf("%d",&n);

    printf("Enter the cost adjacency matrix:\n");
    for(i=0; i<n; i++){
        for(j=0; j<n; j++){
            scanf("%d", &cost[i][j]);
        }
    }
    prims(cost,n);
    printf("Edges of the minimal spanning tree:\n");
    for(i=0; i<n-1; i++)
        printf("(%d,%d)", t[i][0], t[i][1]);
    printf("\n Sum of minimal spanning tree %d\n",
        sum);
    return 0;
}
void prims(int cost[10][10], int n){
    int i, j, u, v;
    int min, source;
    int p[10], d[10], s[10];
```

__/__/____

```
min=999;
source = 0;
for(i=0; i<n ; i++) {  // Initialize Array
    d[i] = cost[source][i];
    s[i] = 0;
}
s[source] = 1;
sum = 0;
int k = 0;
// Find MST
for(i=0; i<n-1; i++){
    min = 999;
    u = -1;
//Find vertex with minimum distance to the MST
    for(j=0; j<n; j++){
        if(s[j] == 0 && d[j] < min) {
            min = d[j];
            u = j;
        }
    }
    if(u != -1){
        t[k][0] = u; // Add edge to MST
        t[k][1] = p[u];
        k++;
        sum += cost[u][p[u]];
        s[u] = 1;
    // update distance
    for(v=0; v<n; v++){
        if(s[v] == 0 && cost[u][v] < d[v]){
            d[v] = cost[u][v];
            p[v] = u;
        }
    }
}}
```

__/__/__

OUTPUT:

Enter the number of vertices = 4
Enter the cost adjacency matrix:

	0	1	5	2
1	0	9999	9999	
5	9999	0	3	
2	9999	3	0	

Edges of the minimal spanning tree:

(1,0) (3,0) (2,3)

Sum of minimal spanning tree: 6.

__/__/____

2) Kruskals Algorithm:

```
#include<stdio.h>
int cost[10][10], n, t[10][2], sum;
void Kruskal(int cost[10][10], int n){
    int min, u, v, count, k;
    int parent[10];
    k=0;
    sum = 0;
    // initialize parent array
    for(int i=0 ; i<n ; i++)
        parent[i] = i;
    count = 0;
    while(count<n-1){
        min = 999;
        u = -1;
        v = -1;
        // find minimum edge
        for(int i=0; i<n; i++){
            for(int j= 0; j<n; j++){
                if(find(parent,i) != find(parent,j)
                    && cost[i][j] < min){
                    min = cost[i][j];
                    u = i;
                    v = j;
                }
            }
        }
        int root_u = find(parent, u);
        int root_v = find(parent, v);
        if(root_u != root_v){
            parent[root_u] = root_v;
            t[k][0] = u;
```

```
            t[k][1] = v;
            sum += min;
            k++;
            count++;
        }
    }
}
int find(int parent[20], int i){
    while(parent[i] != i){
        i = parent[i];
    }
    return i;
}
int main(){
    int i, j;
    printf("Enter the no. of vertices:");
    scanf("%d", &n);

    printf("Enter the cost adjacency matrix:\n");
    for(i=0; i<n; i++){
        for(j=0; j<n; j++)
            scanf("%d", &cost[i][j]);
    }
    Kruskal(cost, n);
    printf("Edges of the minimal spanning tree:\n");

    for(i=0; i<n-1; i++)
        printf("(%d,%d)", t[i][0], t[i][1]);
    printf("\n Sum of minimal spanning tree: %d\n",
        sum);
    return 0;
}
```

__/__/____

OUTPUT:

Enter the number of vertices : 4
Enter the cost adjacency matrix :

0	1	5	2
1	0	9999	9999
5	9999	0	3
2	9999	3	0

Edges of the minimal spanning tree :
(0,1) (0,3) (2,3)
Sum of minimal spanning tree : 6

3) Dijkstra's Algorithm:

```
#include<stdio.h>
int cost[10][10], n, result[10][2], weight[10];
void dijkstra(int cost[][10], int s){
  int d[10], p[10], visited[10];
  int i, j, min, v, u, k;
  for(i=0; i<10; i++){
      d[i] = 999;
      visited[i] = 0;
      p[i] = s;
  }
  d[s] = 0;
  visited[s] = 1;
  for(i=0; i<n; i++){
     min = 999;
     u = 0;
     for(j=0; j<n; j++){
        if(visited[j] == 0){
           if(d[j] < min){
              min = d[j];
              u = j;
           }
        }
     }
     visited[u] = 1;
     for(v=0; v<n; v++){
        if(visited[v] == 0 && (d[u] + cost[u][v]
           < d[v])){
           d[v] = d[u] + cost[u][v];
           p[v] = u;
        }
     }
  }
```

```
for( i=0; i<n; i++) {
    result[i][0] = p[i];
    result[i][1] = i;
    weight[i] = d[i];
    }
}
int main() {
    int i, j, s;
    printf("Enter the no. of vertices : ");
    scanf("%d",&n);
    printf("Enter the adjacency matrix :\n");
    for(i=0; i<n; i++){
        for(j=0; j<n; j++) {
            scanf("%d", &cost[i][j]);
        }
    }
    printf("Enter the source vertex : ");
    scanf("%d", &s);
    dijkstras(cost, s);
    printf("Path :\n");
    for(i=1; i<n; i++)
        printf("(%d,%d) with the weight %d",
            result[i][0], result[i][1],
            weight[result[i][1]]);
    }
return 0;
}
```

P.T.O

OUTPUT

__/__/__

OUTPUT:

Enter the number of vertices: 4

Enter the cost adjacency matrix:

P.T.O

__/__/____

LAB-6

→ Fractional Knapsack

```
#include<stdio.h>
void knapsack(int n, int p[], int w[], int W)
{
  int used[n];
  for(int i=0; i<n; i++)
      used[i]=0;
  int cur_w = W;
  float tot_v=0.0;

  while(cur_w>0){
      maxi=-1;
      for(i=0;i<n;i++){
        if((used[i]==0)&&((maxi==-1)
          ||((float)w[maxi]/p[maxi]
          maxi=i;
      used[maxi]=1;
      if(w[maxi]<=cur_w){
          cur_w-=w[maxi];
          cur_w=0;
          tot_v+=(float)taken/p
          [maxi]*p[maxi];
      }
    printf("Filled the bag with objects
        %.2f\n",tot_v);
}
int main(){
    int n, w;
    printf("Enter the weight of the obj
        -ects:");
    for(int i=0; i<n; i++)
        scanf("%d",&w);
```

__/__/__

```
Knapsack (n, p, w, W);
return 0;
}
```

OUTPUT:

Enter the weight of the objects 1 3 5 4 7
3 2

Filled the bag with objects worth 36.00.

__/__/____

LAB - 7

→ N-queens Problem:

```
#include <stdio.h>
#include <stdbool.h>

bool place(int [], int);
void printf(int[], int);
void nQueens(int);
int main(){
    int n;
    printf("Enter the number of
            queens: ");
    scanf("%d", &n);
    nQueens( n);
    return 0;
}
void nQueens(int n){
    int x[10];
    int count =0;
    int k= 1;

    while(k!=0){
      x[k] = x[k]+1;
      while(x[k]<=n && !place(x,k)){
            x[k]= x[k]+1;
      }
      if( x[k] <=n){
            k++;
            x[k]=0;
      }
      else
          k--;
}
```

```
    printf("Total solns : %d\n", count);
}

void printSC(int x[10], int n){

    for(int p=1; p<=n; p++)
        printf("%d ", x[p]);
    }
    printf("\n");
}
```

OUTPUT:

```
Enter the number of queens : 4
2 4 1 3
Solution found
3 1 4 2
Solution found

Total Solns : 2.
```

15/7/24